

# SPECIFICATION

Electronic Version 1.2.8

Stylesheet Version 1.0

## METHOD AND SYSTEM FOR ADVANCED RESTART OF APPLICATION SERVERS PROCESSING TIME-CRITICAL REQUESTS

### Background of the Invention

#### [0001] FIELD OF THE INVENTION

[0002] The invention relates to a data processing environment consisting of one or more application servers processing requests, particularly time-critical requests, on behalf of at least one application and, more specifically, to a method and a system for controlling restart processing after termination of one or more resource managers that are involved in managing resources needed by an application server to fulfill application requests.

#### [0003] BACKGROUND OF THE INVENTION

[0004] A typical data processing environment consisting of an application server for processing time-critical transactions is shown in FIG. 1. An application server implements a collection of related services which are requested from applications via application clients. Applications, application servers, and application clients may be implemented by computer programs of any nature not limited to a specific type of implementation. Application, application server, and application client are only logical structures; for example, an application server could be the application that requests services from another application server or even requests from itself.

[0005] Application servers typically run on the second tier of a three-tier system

structure. In such a three-tier system structure the application client is on the first tier, requesting services from an application server on the second tier, which in turn requests services from back-end applications that are located on the third tier. The three-tier system structure is a conceptual structure. That means, even if it is typical to deploy the different tiers onto different computers, there is no need to do so; the deployment of the complete structure onto a single computer is a completely valid approach. It should be noted that a three-tier structure is just a special case of an n-tier structure. For example, if users are interacting via a Web browser, then the Web browser would be considered running on tier 0 and the associated web server which runs the application requesting services from the application server is considered running on tier 1.

- [0006] Application servers are typically stateless, that is, they store any necessary state information into a persistent data storage. State information is data that the application server maintains between two subsequent requests from a client so that the individual requests from the application requesting services can be correlated.
- [0007] Application servers, in general, run the processing of requests as transactions in the sense of ACID transactions which means all requests are either processed completely or not at all. A thorough representation of transaction technology is given by Jim Gray and Andreas Reuter, "Transaction Processing: Concepts and Techniques", Morgan Kaufmann Publishers, Inc., 1993. In the case of an error, the transaction is aborted, all resources are reset to the state they have been before the transaction was started. For example, if the access to the relational database that maintains the application server state fails, all changes made to any of the involved resources are backed out, ie. the message in a message queue that represents the request is put back into the queue from where it was read. This requires that all resources that are used by the application server for persistent data are managed by resource managers that can participate in transactions.
- [0008] If any one of the involved software components fails, all components must be brought back to the state when the failure of the one or more components occurred so that processing can continue. The process of bringing a failed component back is called a restart process. The time it takes to restart a component depends on how

much work needs to be undone and redone to re-establish the state of the component at the time of failure.

[0009] Resource managers maintain a log that contains entries about all changes that were applied to their managed resource. The log is typically maintained on highly available disks that, for example, exploit RAID technology. In addition, the resource managers periodically take snapshots of their current state and write this information as a checkpoint into the log. This information is used for recovery from resource manager failures, such as the abnormal termination of the resource manager or the loss of the media on which the resource manager maintains the resource. Recovering from an abnormal termination of the resource manager is called crash recovery, while recovering from a media failure is called media recovery. It should be noted that media recovery is no longer considered an issue in resource manager recovery due to the high availability of the disks on which the data is maintained and is therefore not considered in the present invention. The component of resource managers that is responsible for recovering from failures is typically called the restart manager. When the resource manager is restarted after the crash, the restart manager uses the checkpoint information and the individual log entries to reconstruct the latest state; that means the restart component needs to figure out which transactions need to be aborted, such as those executing when the crash occurred, and which committed updates need to be redone. This is done by locating the checkpoint record and processing all log records that are coming after the checkpoint record. Thus, the frequency with which checkpoint records are taken, determines the amount of processing that is required to restart the failing component after a failure.

[0010] In the prior art approaches, different resource managers have different policies when to write a checkpoint record. These policies are typically specified via some global settings, such as the number of processed messages in a message queuing system or the time period between two subsequent checkpoints in a relational database management system. These settings may be fixed so that they can not be changed by the user, as for example in a message queuing system MQSeries where the number of processed messages is set to 1000; or these settings may be variable, as for example in a relational database management system DB2, where the time between checkpoints can be set by the user.



- [0011] A resource manager typically serves multiple applications such as multiple different application servers. In this case, the log is used for all changes caused by all applications. However, some resource managers, such as DB2, allow multiple instances to run. An instance consists of all the information needed to support users; it has its own catalog for managing metadata, own log, and appropriate user data. A particular application server can be associated with a particular instance. In this case, the log is only used for the operations of the particular application server.
- [0012] The checkpoint record frequency settings for each of the involved resource managers determines how long it will take to recover from the crash of one or more of the resource managers. As each participating resource manager takes checkpoints independently, the maximum restart time of the application server can be calculated as the restart time of the resource manager with the longest restart time plus the restart time of the application server itself.
- [0013] Typically, the restart time is obtained by running simulations with the application server. Since simulation never matches real-life situations, the estimated restart time is only an approximate value.
- [0014] Taking checkpoints is not only a time-consuming operation, it also slows down processing of requests as the resource manager must dedicate processing and I/O resources for taking the checkpoint. Since there is no correlation between the processing of the application server and the frequency of checkpoint taking by the involved resource managers, checkpoints may be taken, even if not required.

## Brief Summary of the Invention

- [0015] It is therefore an aspect of the present invention to provide a method and a system which guarantee that in the case of a failure of the environment, such as a crash of one or more of the resource managers, the request processing time of the application server does not exceed a pre-specified, in particular a user-defined, request processing time.
- [0016] It is another aspect of the present invention to provide such a method and a system which enable an application server that processes time-critical requests to guarantee that a user-specified request processing time is not exceeded, even if any

of the resource managers needed to satisfy the requests terminates abnormally.

[0017] It is yet another aspect of the present invention to provide such a method and a system which minimize the number of checkpoints being taken by the resource managers involved in running the application server transactions.

[0018] The prior art problems are solved by the method according to the invention by having the resource managers taking checkpoints in a frequency that guarantees that the restart of the application server, including the restart of the resource managers and including the processing time of the request, does not exceed the specified request processing time. Even in the case of failure of any of the involved resource managers, it is achieved that the time it takes to have the requests processed again is shorter than a specified request processing time. Thus the proposed mechanism ensures that the resource managers take checkpoints in such a frequency that the time it takes to have the application server processing requests again is shorter than the specified request processing time.

[0019] The concept underlying the invention is based on the observation that the frequency with which checkpoint records are taken determines the amount of processing that is required to restart a failing component after a failure.

[0020] The method according to the invention preferably specifies two mechanisms of controlling when a checkpoint needs to be written, one being a load-controlled mechanism and the other being a restart-time-controlled mechanism.

[0021] In both cases, specification of the request processing time may either be hard coded in the application server or provided to the application server as a parameter. As an additional option, the application server may support changing of the request processing time during operation, for example, by offering an appropriate application programming interface, or some graphical/textual interface.

[0022] In the load-controlled method, the application server knows the amount of data that each resource manager writes into its log and the amount of time it takes the resource manager to process the log in the case of a restart. Thus, the application server knows at any time the currently necessary restart time of each resource manager and, based on that information, requests the taking of checkpoints from the

resource managers so that the specified request processing time could even be met in the case of a failure of one or more resource managers. The load-controlled method is based on the assumption that the resource managers provide the capability for calling applications to request the taking of checkpoints.

- [0023] In the restart-time-controlled method, the resource manager provides the capability for the specification of a restart time. This allows the application server to determine from the user specified request processing time the appropriate restart time for each of the involved resource managers. This information is handed over to the resource manager when the application server connects to the resource manager the first time or when the user specified request processing time is changed.
- [0024] Thereupon, a further advantage of the proposed mechanism is that checkpoints have only to be taken if necessary, resulting in an improved throughput of the application server.

### Brief Description of the Several Views of the Drawings

- [0025] The invention will now be described in the following in more detail by way of embodiments whereby reference is made to the accompanying drawings. In the drawings,
- [0026] FIG. 1 is a schematic block diagram depicting an application server according to the prior art where the present invention can be applied;
- [0027] FIG. 2 is a schematic block diagram depicting the internal structure and transaction boundaries of a message-based application server where the present invention can be applied;
- [0028] FIG. 3 is a flow diagram illustrating the user controlled resource manager checkpoint controlling mechanism according to the present invention;
- [0029] FIG. 4 is a flow diagram illustrating detailed method steps of the load-controlled checkpoint mechanism according to the present invention; and
- [0030] FIG. 5 shows exemplary actions performed by an application server where a resource manager supports a user-defined restart time.

## Detailed Description of the Invention

[0031] DESCRIPTION OF THE PREFERRED EMBODIMENTS

[0032] FIG. 1 shows the typical structure of an application server. A number of applications request services from the application server via application clients. Application server 100 implements a collection of services which are requested from applications 110, 112, and 114 via application clients 120, 122, and 124. Applications 110, 112, and 114, application server 100, and application clients 120, 122, and 124 may be implemented by computer programs of any nature not limited to any specific type or implementation. Applications 110, 112, and 114, application server 100, and application clients 120, 122, and 124 are only logical structures. For example, application server 100 can be application 110 itself that requests services from another application server (not shown here) or even requests services from itself. As application server 100 is stateless, all information that is needed between subsequent requests from applications 110, 112, and 114 are kept in data store 130.

[0033] FIG. 2 consists of a particular implementation of an application server, namely message-based application server 200, which means the communication between application client 210 and application server 200 is based on asynchronous reliable message exchange. But it is understood hereby that the invention is not limited to a certain communication paradigm. FIG. 2 is used to illustrate how an application server processes requests as transactions; the transaction boundaries are indicated by dashed lines.

[0034] When application 220 requests a service from application server 200, application client 210 puts a message reflecting the request of application 220 into application server input queue 280. This is done as transaction 260, labeled Transaction 1, so that when the request has been successfully put into application server input queue 280, the request of application 220 will be honored.

[0035] Application server 200 processes requests within transaction 240, labeled Transaction 2. The transaction consists of reading the request from input queue 280, performing processing based on the request, saving processing state information persistent into data store 230, and putting a response into application server output queue 285.

[0036] Application 220 gets this response by application client 210 reading the response from application server output queue 285. This process is run as transaction 250, labeled transaction 3.

[0037] In the processing of a request and generating a response by application server 200, multiple resources are accessed and possibly modified. Since multiple resources and thus multiple resource managers are involved in such a transaction, the transaction must be coordinated via transaction manager 270 typically using a two-phase-commit (2PC) protocol. The individual resources are typically managed by different resource managers. For example, the application server's input and output queue 280, 285 are managed by a message queuing system (not shown) and data store 230 by a relational database management system (not shown). The involvement of multiple resource managers mandates a transaction manager which coordinates the processing of the individual resource managers when the transaction needs to be committed or aborted. The 2PC protocol is the protocol that transaction managers typically exploit to coordinate commit/abort processing with the resource managers involved in a particular transaction. Further information can be found in the textbook "Transaction Processing: Concepts and Techniques", authored by Jim Gray and Andreas Reuter and published by Morgan Kaufmann Publishers, Inc., 1993 which is regarded to be entirely incorporated herein by reference.

[0038] Referring again to FIG. 2, application server 200 processes each individual client request as a transaction. The term transaction should not be understood as limited to the classical definition of transactions, that means having ACID properties, but to all units of work concepts that provide some form of transaction control by weakening one or more of the ACID properties.

[0039] It is further noted that, although the above description is related to an application server, the scope of the present invention is by no means limited to such an application server and, moreover, can be used for any other application which performs the described interaction patterns with resource managers.

[0040] FIG. 3 shows the processing of a request by the resource manager (RM) when the resource manager supports user-specified checkpoint processing based on the present invention. It assumes that one or a multitude of users has specified a

maximum restart time, hereafter referred to as restart time, for the resource manager and the resource manager has stored this information internally. It is noted that the user or multitude of users, alternatively, can specify two or more restart times wherein the proposed mechanism takes the minimum restart time for generating checkpoints. When the resource manager processes next request 300, it first determines the current settings for user specified restart time 310. This restart time must not be exceeded in case of failure. Next, the resource manager determines the restart time needed for current request 320 and the potential previous requests, which would participate in restart processing since the last checkpoint processing, and then calculates new restart time 330. If the new restart time would exceed the specified restart time effect of the request on overall restart time 350, a checkpoint is taken 340. Then the request is processed 360 and the new restart time is calculated 370. It should be noted, that FIG. 3 is for illustration purpose only; actual implementations are most likely more sophisticated.

- [0041] In the following, two different embodiments of the invention are described in more detail.
- [0042] Load-controlled checkpointing
- [0043] It is hereby assumed that resource managers provide the capability for applications to request checkpoints. How the resource managers externalize this capability is not relevant, whether it is the sending of a message to queue or the invocation of the request via an application programming interface.
- [0044] FIG. 4 shows the processing of a request by the application server (AS) using load-controlled checkpointing according to the present invention. It is assumed that the application server has knowledge about the amount of log data written by each resource manager and the amount of time it takes to process the log during recovery. It further assumes that the user has specified the maximum request response time (not the restart time), hereafter referred to as response time, for the application server, and the application server has stored this information internally.
- [0045] When the application server processes next request 400, it first determines the current settings for user specified request response time 410. This request response

time must not be exceeded in case of failure. Next, the application server performs a set of actions (indicated by loop 470) for all resource managers. First, the application server determines the restart time needed for current request 420 and the potential previous requests, which would participate in restart processing since the last checkpoint processing, and then calculates new restart time 430. If the new restart time would exceed the restart time necessary to keep within requested response time 450, the resource manager is requested to take checkpoint 440. The mapping of the request response time to the restart time takes into effect additional processing that is associated with the restart after a failure, such as starting the application server or reattaching to the failing resource manager. After all resource managers have been processed, the request is processed 460 and the new restart time is calculated for all resource managers 470. It should be noted, that FIG. 4 is for illustration purpose only; actual implementations are most likely more sophisticated.

- [0046] If the resource manager does not support the notion of a resource manager instance, then the resource manager must provide the capability to assign a separate log to a particular application. In addition, the resource manager must provide the capability that applications can request that checkpoints are taken only for a particular log. When a crash occurs, then the resource manager must process this log before any other log, unless there is no time penalty when processing multiple logs in parallel.
- [0047] To summarize, in the load-controlled checkpointing approach the above teaching is embodied within an application server which controls the checkpointing frequency based upon guaranteed response time requirements on behalf of the underlying resource managers. In addition, this approach reflects the processing required to restart the application server itself.
- [0048] Restart time controlled checkpointing
- [0049] The load-controlled checkpoint approach requires that the application server has a deep understanding of the logging and restart operations of each of the involved resource managers. In particular, the metrics associated with logging and restart need to be changed whenever the resource manager is changed.
- [0050] In order to increase performance and throughput of the application server(s), in

the present embodiment, the resource managers themselves keep track of the restart time instead of the application server. The resource manager externalizes the capability for applications to set the restart time and the resource manager then takes a checkpoint whenever the specified restart time is reached. FIG. 3 shows how a resource manager could implement this capability.

[0051] FIG. 5 shows the actions that the application server needs to take when the resource manager (RM) supports a user-defined restart time. In step 500, the application server obtains user specified maximum request response time, hereafter referred to as request response time. The application server then performs a set of actions (indicated by a loop) for all resource managers. Step 510 calculates the appropriate restart time for the resource manager from the specified request response time. Transformation of the request response time to the restart time of the resource manager is necessary to cope with additional processing needs, such as the start up of the application server itself. In step 520, the application server hands over this restart time to the resource manager. The resource manager itself would then execute the code shown in FIG. 3 if called by the application server. It should be noted, that FIG. 5 is for illustration purpose only; actual implementations are most likely more sophisticated.

[0052] Although specific embodiments of the present invention have been illustrated in the accompanying drawings and described in the foregoing detailed description, it will be understood that the invention is not limited to the particular embodiments described herein, but is capable of numerous rearrangements, modifications and substitutions without departing from the scope of the invention. The following claims are intended to encompass all such modifications.